GALAHAD LSQP

USER DOCUMENTATION GALAHAD Optimization Library version 5.1

1 SUMMARY

This package uses an primal-dual interior-point method to solve the **linear** or **separable convex quadratic programming problem**

$$\text{minimize } \tfrac{1}{2}\sum_{i=1}^{n} w_i^2(x_i - x_i^0)^2 + \mathbf{g}^T\mathbf{x} + f$$

subject to the general linear constraints

$$c_i^l \leq \mathbf{a}_i^T\mathbf{x} \leq c_i^u, \quad i = 1,\ldots,m,$$

and the simple bound constraints

$$x_j^l \leq x_j \leq x_j^u, \quad j = 1,\ldots,n,$$

where the vectors $\mathbf{w}$, $\mathbf{g}$, $\mathbf{x}^0$, $\mathbf{a}_i$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$, $\mathbf{x}^u$ and the scalar f are given. Full advantage is taken of any zero coefficients in the vectors $\mathbf{a}_i$. Any of the constraint bounds c_i^l, c_i^u, x_j^l and x_j^u may be infinite. In the special case where $\mathbf{w} = 0$, $\mathbf{g} = 0$ and $f = 0$, the so-called analytic center of the feasible set will be found, while linear programming, or constrained least distance, problems may be solved by picking $\mathbf{w} = 0$, or $\mathbf{g} = 0$ and $f = 0$, respectively.

The more-modern GALAHAD package `CQP` offers similar functionality, and is often to be preferred.

ATTRIBUTES — Versions: `GALAHAD_LSQP_single, GALAHAD_LSQP_double`. **Uses:** `GALAHAD_CLOCK`, `GALAHAD_SYMBOLS`, `GALAHAD_SPACE`, `GALAHAD_TOOLS`, `GALAHAD_SPECFILE`, `GALAHAD_SMT`, `GALAHAD_QPT`, `GALAHAD_QPP`, `GALAHAD_QPD`, `GALAHAD_ROOTS`, `GALAHAD_SBLS`, `GALAHAD_FDC`. **Date:** October 2001. **Origin:** N. I. M. Gould, Rutherford Appleton Laboratory, and Ph. L. Toint, University of Namur, Belgium. **Language:** Fortran 95 + TR 15581 or Fortran 2003. **Parallelism:** Some options may use OpenMP and its runtime library.

2 HOW TO USE THE PACKAGE

The package is available with single, double and (if available) quadruple precision reals, and either 32-bit or 64-bit integers. Access to the 32-bit integer, single precision version requires the `USE` statement

```
USE GALAHAD_LSQP_single
```

with the obvious substitution `GALAHAD_LSQP_double`, `GALAHAD_LSQP_quadruple`, `GALAHAD_LSQP_single_64`, `GALAHAD_LSQP_double_64` and `GALAHAD_LSQP_quadruple_64` for the other variants.

If it is required to use more than one of the modules at the same time, the derived types `SMT_type`, `QPT_problem_type`, `LSQP_time_type`, `LSQP_control_type`, `LSQP_inform_type` and `LSQP_data_type` (Section 2.4) and the subroutines `LSQP_initialize`, `LSQP_solve`, `LSQP_terminate`, (Section 2.5) and `LSQP_read_specfile` (Section 2.7) must be renamed on one of the `USE` statements.

2.1 Matrix storage formats

The constraint Jacobian $\mathbf{A}$, that is, the matrix whose rows are the vectors $\mathbf{a}_i^T$, $i = 1,\ldots,m$, may be stored in one of three input formats.

2.1.1 Dense storage format

The matrix is stored as a compact dense matrix by rows, that is, the values of the entries of each row in turn are stored in order within an appropriate real one-dimensional array. Component $n*(i-1)+j$ of the storage array A%val will hold the value a_{ij} for $i = 1,\ldots,m$, $j = 1,\ldots,n$.

2.1.2 Sparse co-ordinate storage format

Only the nonzero entries of the matrix are stored. For the l-th entry of $\mathbf{A}$, its row index i, column index j and value a_{ij} are stored in the l-th components of the integer arrays A%row, A%col and real array A%val, respectively. The order is unimportant, but the total number of entries A%ne is also required.

2.1.3 Sparse row-wise storage format

Again only the nonzero entries are stored, but this time they are ordered so that those in row i appear directly before those in row $i+1$. For the i-th row of $\mathbf{A}$, the i-th component of a integer array A%ptr holds the position of the first entry in this row, while A%ptr $(m+1)$ holds the total number of entries plus one. The column indices j and values a_{ij} of the entries in the i-th row are stored in components $l =$ A%ptr(i), ...,A%ptr $(i+1)-1$ of the integer array A%col, and real array A%val, respectively.

For sparse matrices, this scheme almost always requires less storage than its predecessor.

2.2 Real and integer kinds

We use the terms integer and real to refer to the fortran keywords REAL(rp_) and INTEGER(ip_), where rp_ and ip_ are the relevant kind values for the real and integer types employed by the particular module in use. The former are equivalent to default REAL for the single precision versions, DOUBLE PRECISION for the double precision cases and quadruple-precision if 128-bit reals are available, and correspond to rp_ = real32, rp_ = real64 and rp_ = real128 respectively as defined by the fortran iso_fortran_env module. The latter are default (32-bit) and long (64-bit) integers, and correspond to ip_ = int32 and ip_ = int64, respectively, again from the iso_fortran_env module.

2.3 Parallel usage

OpenMP may be used by the GALAHAD_LSQP package to provide parallelism for some solvers in shared memory environments. See the documentation for the GALAHAD package SLS for more details. To run in parallel, OpenMP must be enabled at compilation time by using the correct compiler flag (usually some variant of -openmp). The number of threads may be controlled at runtime by setting the environment variable OMP_NUM_THREADS.

MPI may also be used by the package to provide parallelism for some solvers in a distributed memory environment. To use this form of parallelism, MPI must be enabled at runtime by using the correct compiler flag (usually some variant of -lmpi). Although the MPI process will be started automatically when required, it should be stopped by the calling program once no further use of this form of parallelism is needed. Typically, this will be via statements of the form

```
CALL MPI_INITIALIZED( flag, ierr )
IF ( flag ) CALL MPI_FINALIZE( ierr )
```

The code may be compiled and run in serial mode.

2.4 The derived data types

Six derived data types are accessible from the package.

2.4.1 The derived data type for holding matrices

The derived data type SMT_TYPE is used to hold the matrix **A**. The components of SMT_TYPE used here are:

m is a scalar component of type INTEGER(ip_), that holds the number of rows in the matrix.

n is a scalar component of type INTEGER(ip_), that holds the number of columns in the matrix.

ne is a scalar variable of type INTEGER(ip_), that holds the number of matrix entries

type is a rank-one allocatable array of type default CHARACTER, that is used to indicate the matrix storage scheme used. Its precise length and content depends on the type of matrix to be stored (see §2.4.2).

val is a rank-one allocatable array of type REAL(rp_) and dimension at least ne, that holds the values of the entries. Each pair of off-diagonal entries $a_{ij} = a_{ji}$ of a matrix **A** is represented as a single entry (see §2.1.1–2.1.3). Any duplicated entries that appear in the sparse co-ordinate or row-wise schemes will be summed.

row is a rank-one allocatable array of type INTEGER(ip_), and dimension at least ne, that may hold the row indices of the entries. (see §2.1.2).

col is a rank-one allocatable array of type INTEGER(ip_), and dimension at least ne, that may hold the column indices of the entries (see §2.1.2–2.1.3).

ptr is a rank-one allocatable array of type INTEGER(ip_), and dimension at least m + 1, that may hold the pointers to the first entry in each row (see §2.1.3).

2.4.2 The derived data type for holding the problem

The derived data type QPT_problem_type is used to hold the problem. The components of QPT_problem_type are:

new_problem_structure is a scalar variable of type default LOGICAL, that is .TRUE. if this is the first (or only) problem in a sequence of problems with identical "structure" to be attempted, and .FALSE. if a previous problem with the same "structure" (but different numerical data) has been solved. Here, the term "structure" refers both to the sparsity patterns of the Jacobian matrices **A** involved (but not their numerical values), to the zero/nonzero/infinity patterns (a bound is either zero, $\pm$ infinity, or a finite but arbitrary nonzero) of each of the constraint bounds, and to the variables and constraints that are fixed (both bounds are the same) or free (the lower and upper bounds are $\pm$ infinity, respectively).

n is a scalar variable of type INTEGER(ip_), that holds the number of optimization variables, n.

m is a scalar variable of type INTEGER(ip_), that holds the number of general linear constraints, m.

Hessian_kind is a scalar variable of type INTEGER(ip_), that is used to indicate whether the weights **w** have special or general values. Possible values for Hessian_kind are:

- 0 In this case, $\mathbf{w} = 0$, and an approximation to the analytic center (if gradient_kind $= 0$, see below) or the solution to the resulting *linear program* (if gradient_kind $\neq 0$) will be computed.
- 1 In this case, $w_i = 1$ for $i = 1, \ldots, n$.
- $\neq$ 0,1 In this case, general values of **w** will be used, and will be provided by the user in the component WEIGHT.

WEIGHT is a rank-one allocatable array type REAL(rp_), that must be allocated to have length n, and its j-th component filled with the value w_i for $i = 1, \ldots, n$, whenever Hessian_kind $\neq 0, 1$. If Hessian_kind = 0, 1, WEIGHT need not be allocated.

X0 is a rank-one allocatable array type REAL(rp_), that must be allocated to have length n, and its j-th component filled with the value x_i^0 for $i = 1,\ldots,n$, whenever Hessian_kind $\neq 0$. If Hessian_kind = 0, X0 need not be allocated.

gradient_kind is a scalar variable of type INTEGER(ip_), that is used to indicate whether the components of the gradient $\mathbf{g}$ have special or general values. Possible values for gradient_kind are:

0 In this case, $\mathbf{g} = 0$.

1 In this case, $g_i = 1$ for $i = 1,\ldots,n$.

$\neq$ 0,1 In this case, general values of $\mathbf{g}$ will be used, and will be provided by the user in the component G.

G is a rank-one allocatable array of dimension n and type REAL(rp_), that holds the gradient $\mathbf{g}$ of the linear term of the quadratic objective function. The j-th component of G, $j = 1,\ldots,n$, contains $\mathbf{g}_j$. If gradient_kind = 0, 1, G need not be allocated.

f is a scalar variable of type REAL(rp_), that holds the constant term, f, in the objective function.

A is scalar variable of type SMT_TYPE that holds the Jacobian matrix $\mathbf{A}$. The following components are used:

A%type is an allocatable array of rank one and type default CHARACTER, that is used to indicate the storage scheme used. If the dense storage scheme (see Section 2.1.1) is used, the first five components of A%type must contain the string DENSE. For the sparse co-ordinate scheme (see Section 2.1.2), the first ten components of A%type must contain the string COORDINATE, while for the sparse row-wise storage scheme (see Section 2.1.3), the first fourteen components of A%type must contain the string SPARSE_BY_ROWS.

For convenience, the procedure SMT_put may be used to allocate sufficient space and insert the required keyword into A%type. For example, if prob is of derived type LSQP_problem_type and involves a Jacobian we wish to store using the sparse row-wise storage scheme, we may simply

```
CALL SMT_put( prob%A%type, 'SPARSE_BY_ROWS' )
```

See the documentation for the GALAHAD package SMT for further details on the use of SMT_put.

A%ne is a scalar variable of type INTEGER(ip_), that holds the number of entries in $\mathbf{A}$ in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be set for either of the other two schemes.

A%val is a rank-one allocatable array of type REAL(rp_), that holds the values of the entries of the Jacobian matrix $\mathbf{A}$ in any of the storage schemes discussed in Section 2.1.

A%row is a rank-one allocatable array of type INTEGER(ip_), that holds the row indices of $\mathbf{A}$ in the sparse co-ordinate storage scheme (see Section 2.1.2). It need not be allocated for either of the other two schemes.

A%col is a rank-one allocatable array variable of type INTEGER(ip_), that holds the column indices of $\mathbf{A}$ in either the sparse co-ordinate (see Section 2.1.2), or the sparse row-wise (see Section 2.1.3) storage scheme. It need not be allocated when the dense storage scheme is used.

A%ptr is a rank-one allocatable array of dimension m+1 and type INTEGER(ip_), that holds the starting position of each row of $\mathbf{A}$, as well as the total number of entries plus one, in the sparse row-wise storage scheme (see Section 2.1.3). It need not be allocated when the other schemes are used.

C_l is a rank-one allocatable array of dimension m and type REAL(rp_), that holds the vector of lower bounds $\mathbf{c}^l$ on the general constraints. The i-th component of C_l, $i = 1,\ldots,m$, contains $\mathbf{c}_i^l$. Infinite bounds are allowed by setting the corresponding components of C_l to any value smaller than -infinity, where infinity is a component of the control array control (see Section 2.4.3).

C_u is a rank-one allocatable array of dimension m and type REAL(rp_), that holds the vector of upper bounds $\mathbf{c}^u$ on the general constraints. The i-th component of C_u, $i = 1,\ldots,m$, contains $\mathbf{c}_i^u$. Infinite bounds are allowed by setting the corresponding components of C_u to any value larger than infinity, where infinity is a component of the control array control (see Section 2.4.3).

X_l is a rank-one allocatable array of dimension n and type REAL(rp_), that holds the vector of lower bounds $\mathbf{x}^l$ on the the variables. The j-th component of X_l, $j = 1,\ldots,n$, contains $\mathbf{x}^l_j$. Infinite bounds are allowed by setting the corresponding components of X_l to any value smaller than -infinity, where infinity is a component of the control array control (see Section 2.4.3).

X_u is a rank-one allocatable array of dimension n and type REAL(rp_), that holds the vector of upper bounds $\mathbf{x}^u$ on the variables. The j-th component of X_u, $j = 1,\ldots,n$, contains $\mathbf{x}^u_j$. Infinite bounds are allowed by setting the corresponding components of X_u to any value larger than that infinity, where infinity is a component of the control array control (see Section 2.4.3).

X is a rank-one allocatable array of dimension n and type REAL(rp_), that holds the values $\mathbf{x}$ of the optimization variables. The j-th component of X, $j = 1,\ldots,n$, contains x_j. The vector $\mathbf{x}^0$ will initially be specified in X.

Z is a rank-one allocatable array of dimension n and type default REAL(rp_), that holds the values $\mathbf{z}$ of estimates of the dual variables corresponding to the simple bound constraints (see Section 4). The j-th component of Z, $j = 1,\ldots,n$, contains z_j.

C is a rank-one allocatable array of dimension m and type default REAL(rp_), that holds the values $\mathbf{Ax}$ of the constraints. The i-th component of C, $i = 1,\ldots,m$, contains $\mathbf{a}_i^T\mathbf{x} \equiv (\mathbf{Ax})_i$.

Y is a rank-one allocatable array of dimension m and type REAL(rp_), that holds the values $\mathbf{y}$ of estimates of the Lagrange multipliers corresponding to the general linear constraints (see Section 4). The i-th component of Y, $i = 1,\ldots,m$, contains y_i.

2.4.3 The derived data type for holding control parameters

The derived data type LSQP_control_type is used to hold controlling data. Default values may be obtained by calling LSQP_initialize (see Section 2.5.1), while components may also be changed by calling GALAHAD_LSQP_read_spec (see Section 2.7.1). The components of LSQP_control_type are:

error is a scalar variable of type INTEGER(ip_), that holds the stream number for error messages. Printing of error messages in LSQP_solve and LSQP_terminate is suppressed if error ≤ 0. The default is error = 6.

out is a scalar variable of type INTEGER(ip_), that holds the stream number for informational messages. Printing of informational messages in LSQP_solve is suppressed if out < 0. The default is out = 6.

print_level is a scalar variable of type INTEGER(ip_), that is used to control the amount of informational output which is required. No informational output will occur if print_level ≤ 0. If print_level $= 1$, a single line of output will be produced for each iteration of the process. If print_level ≥ 2, this output will be increased to provide significant detail of each iteration. The default is print_level = 0.

maxit is a scalar variable of type INTEGER(ip_), that holds the maximum number of iterations which will be allowed in LSQP_solve. The default is maxit = 1000.

start_print is a scalar variable of type INTEGER(ip_), that specifies the first iteration for which printing will occur in LSQP_solve. If start_print is negative, printing will occur from the outset. The default is start_print = -1.

stop_print is a scalar variable of type INTEGER(ip_), that specifies the last iteration for which printing will occur in LSQP_solve. If stop_print is negative, printing will occur once it has been started by start_print. The default is stop_print = -1.

infeas_max is a scalar variable of type INTEGER(ip_), that specifies the number of iterations for which the overall infeasibility of the problem is not reduced by at least a factor reduce_infeas before the problem is flagged as infeasible (see reduce_infeas). The default is infeas_max = 200.

muzero_fixed is a scalar variable of type INTEGER(ip_), that specifies the number of iterations before the initial barrier parameter (see muzero) may be altered. The default is muzero_fixed = 1.

indicator_type is a scalar variable of type INTEGER(ip_), that specifies the type of indicator used to assess when a variable or constraint is active. Possible values are:

1 a variable/constraint is active if and only if the distance to its neaerest bound is no larger than indicator_tol_p (see below).

2 a variable/constraint is active if and only if the distance to its neaerest bound is no larger than indicator_tol_pd (see below) times the magnitude of its corresponding dual variable.

3 a variable/constraint is active if and only if the distance to its neaerest bound is no larger than indicator_tol_tapia (see below) times the distance to the same bound at the previous iteration.

The default is indicator_type = 3.

restore_problem is a scalar variable of type INTEGER(ip_), that specifies how much of the input problem is to be retored on output. Possible values are:

0 nothing is restored.

1 the vector data $\mathbf{w}$, $\mathbf{g}$, $\mathbf{c}^l$, $\mathbf{c}^u$, $\mathbf{x}^l$, and $\mathbf{x}^u$ will be restored to their input values.

2 the entire problem, that is the above vector data along with the Jacobian matrix $\mathbf{A}$, will be restored.

The default is restore_problem = 2.

infinity is a scalar variable of type REAL(rp_), that is used to specify which constraint bounds are infinite. Any bound larger than infinity in modulus will be regarded as infinite. The default is infinity = 10^{19}.

stop_p is a scalar variable of type REAL(rp_), that holds the required accuracy for the primal infeasibility (see Section 4). The default is stop_p = $u^{1/3}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_LSQP_double).

stop_d is a scalar variable of type default REAL(rp_), that holds the required accuracy for the dual infeasibility (see Section 4). The default is stop_d = $u^{1/3}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_LSQP_double).

stop_c is a scalar variable of type default REAL(rp_), that holds the required accuracy for the violation of complementarity slackness (see Section 4). The default is stop_c = $u^{1/3}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_LSQP_double).

prfeas is a scalar variable of type REAL(rp_), that aims to specify the closest that any initial variable may be to infeasibility. Any variable closer to infeasibility than prfeas will be moved to prfeas from the offending bound. However, if a variable is range bounded, and its bounds are closer than prfeas apart, it will be moved to the mid-point of the two bounds. The default is prfeas = 1.0.

dufeas is a scalar variable of type REAL(rp_), that aims to specify the closest that any initial dual variable or Lagrange multiplier may be to infeasibility. Any variable closer to infeasibility than prfeas will be moved to dufeas from the offending bound. However, if a dual variable is range bounded, and its bounds are closer than dufeas apart, it will be moved to the mid-point of the two bounds. The default is dufeas = 1.0.

muzero is a scalar variable of type REAL(rp_), that holds the initial value of the barrier parameter. If muzero is not positive, it will be reset automatically to an appropriate value. The default is muzero = -1.0.

reduce_infeas is a scalar variable of type default REAL(rp_), that specifies the least factor by which the overall infeasibility of the problem must be reduced, over infeas_max consecutive iterations, for it not be declared infeasible (see infeas_max). The default is reduce_infeas = 0.99.

potential_unbounded is a scalar variable of type default REAL(rp_), that specifies smallest value of the potential function divided by the number of one-sided variable and constraint bounds that will be tolerated before the analytic center is declared to be unbounded. The default is potential_unbounded = -10.0.

identical_bounds_tol is a scalar variable of type REAL(rp_). Every pair of constraint bounds (c_i^l, c_i^u) or (x_j^l, x_j^u) that is closer than identical_bounds_tol will be reset to the average of their values, $\frac{1}{2}(c_i^l + c_i^u)$ or $\frac{1}{2}(x_j^l + x_j^u)$ respectively. The default is identical_bounds_tol = u, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_LSQP_double).

indicator_tol_p is a scalar variable of type REAL(rp_) that provides the indicator tolerance associated with the test indicator_type = 1. The default is indicator_tol_p = $u^{1/3}$, where u is EPSILON(1.0) (EPSILON(1.0D0) in GALAHAD_LSQP_double).

indicator_tol_pd is a scalar variable of type REAL(rp_) that provides the indicator tolerance associated with the test indicator_type = 2. The default is indicator_tol_pd = 1.0.

indicator_tol_tapia is a scalar variable of type REAL(rp_) that provides the indicator tolerance associated with the test indicator_type = 3. The default is indicator_tol_tapia = 0.9.

cpu_time_limit is a scalar variable of type REAL(rp_), that is used to specify the maximum permitted CPU time. Any negative value indicates no limit will be imposed. The default is cpu_time_limit = - 1.0.

clock_time_limit is a scalar variable of type REAL(rp_), that is used to specify the maximum permitted elapsed system clock time. Any negative value indicates no limit will be imposed. The default is clock_time_limit = - 1.0.

remove_dependencies is a scalar variable of type default LOGICAL, that must be set .TRUE. if the algorithm is to attempt to remove any linearly dependent constraints before solving the problem, and .FALSE. otherwise. We recommend removing linearly dependencies. The default is remove_dependencies = .TRUE..

treat_zero_bounds_as_general is a scalar variable of type default LOGICAL. If it is set to .FALSE., variables which are only bounded on one side, and whose bound is zero, will be recognised as non-negativities/non-positivities rather than simply as lower- or upper-bounded variables. If it is set to .TRUE., any variable bound x_j^l or x_j^u which has the value 0.0 will be treated as if it had a general value. Setting treat_zero_bounds_as_general to .TRUE. has the advantage that if a sequence of problems are reordered, then bounds which are "accidentally" zero will be considered to have the same structure as those which are nonzero. However, GALAHAD_LSQP is able to take special advantage of non-negativities/non-positivities, so if a single problem, or if a sequence of problems whose bound structure is known not to change, is/are to be solved, it will pay to set the variable to .FALSE.. The default is treat_zero_bounds_as_general = .FALSE..

just_feasible is a scalar variable of type default LOGICAL, that must be set .TRUE. if the algorithm should stop as soon as a feasible point of the constraint set is found, and .FALSE. otherwise. The default is just_feasible = .FALSE..

getdua is a scalar variable of type default LOGICAL, that must be set .TRUE. if the user-provided estimates of the dual variables should be replaced by estimates whose aim is to try to balance the requirements of dual feasibility and complementary slackness, and .FALSE. if users estimates are to be used. The default is getdua = .FALSE..

feasol is a scalar variable of type default LOGICAL, that should be set .TRUE. if the final solution obtained will be perturbed so that variables close to their bounds are moved onto these bounds, and .FALSE. otherwise. The default is feasol = .FALSE..

balance_initial_complentarity is a scalar variable of type default LOGICAL, that should be set .TRUE. if the initial complementarity is required to be balanced, and .FALSE. otherwise. The default is balance_initial_complentarity = .FALSE..

prefix is a scalar variable of type default CHARACTER and length 30, that may be used to provide a user-selected character string to preface every line of printed output. Specifically, each line of output will be prefaced by the string prefix(2:LEN(TRIM(prefix))-1), thus ignoring the first and last non-null components of the supplied string. If the user does not want to preface lines by such a string, they may use the default prefix = "".

FDC_control is a scalar variable of type FDC_control_type whose components are used to control any detection of linear dependencies performed by the package GALAHAD_FDC. See the specification sheet for the package GALAHAD_FDC for details, and appropriate default values.

SBLS_control is a scalar variable of type SBLS_control_type whose components are used to control factorizations performed by the package GALAHAD_SBLS. See the specification sheet for the package GALAHAD_SBLS for details, and appropriate default values.

2.4.4 The derived data type for holding timing information

The derived data type LSQP_time_type is used to hold elapsed CPU and system clock times for the various parts of the calculation. The components of LSQP_time_type are:

total is a scalar variable of type REAL(rp_), that gives the total CPU time spent in the package.

preprocess is a scalar variable of type REAL(rp_), that gives the CPU time spent reordering the problem to standard form prior to solution.

find_dependent is a scalar variable of type REAL(rp_), that gives the CPU time spent detecting and removing linearly-dependent equality constraints

analyse is a scalar variable of type REAL(rp_), that gives the CPU time spent analysing the required matrices prior to factorization.

factorize is a scalar variable of type REAL(rp_), that gives the CPU time spent factorizing the required matrices.

solve is a scalar variable of type REAL(rp_), that gives the CPU time spent computing the search direction.

clock_total is a scalar variable of type REAL(rp_), that gives the total elapsed system clock time spent in the package.

clock_preprocess is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent reordering the problem to standard form prior to solution.

clock_find_dependent is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent detecting and removing linearly-dependent equality constraints

clock_analyse is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent analysing the required matrices prior to factorization.

clock_factorize is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent factorizing the required matrices.

clock_solve is a scalar variable of type REAL(rp_), that gives the elapsed system clock time spent computing the search direction.

2.4.5 The derived data type for holding informational parameters

The derived data type LSQP_inform_type is used to hold parameters that give information about the progress and needs of the algorithm. The components of LSQP_inform_type are:

status is a scalar variable of type INTEGER(ip_), that gives the exit status of the algorithm. See Section 2.6 for details.

alloc_status is a scalar variable of type INTEGER(ip_), that gives the status of the last attempted array allocation or deallocation. This will be 0 if status = 0.

bad_alloc is a scalar variable of type default CHARACTER and length 80, that gives the name of the last internal array for which there were allocation or deallocation errors. This will be the null string if status = 0.

iter is a scalar variable of type INTEGER(ip_), that gives the total number of iterations required.

factorization_status is a scalar variable of type INTEGER(ip_), that gives the return status from the matrix factorization.

factorization_integer is a scalar variable of type long INTEGER(ip_), that gives the amount of integer storage used for the matrix factorization.

factorization_real is a scalar variable of type INTEGER(int64), that gives the amount of real storage used for the matrix factorization.

nfacts is a scalar variable of type INTEGER(ip_), that gives the total number of factorizations performed.

nbacts is a scalar variable of type INTEGER(ip_), that gives the total number of backtracks performed during the sequence of linesearches.

obj is a scalar variable of type REAL(rp_), that holds the value of the objective function at the best estimate of the solution found.

potential is a scalar variable of type REAL(rp_), that holds the value of the potential function at the best estimate of the analytic center found in the special case when $\mathbf{w} = 0$.

non_negligible_pivot is a scalar variable of type REAL(rp_), that holds the value of the smallest pivot larger than control%zero_pivot when searching for dependent linear constraints. If non_negligible_pivot is close to control%zero_pivot, this may indicate that there are further dependent constraints, and it may be worth increasing control%zero_pivot above non_negligible_pivot and solving again.

feasible is a scalar variable of type default LOGICAL, that has the value .TRUE. if the output value of $\mathbf{x}$ satisfies the constraints, and the value .FALSE. otherwise.

time is a scalar variable of type LSQP_time_type whose components are used to hold elapsed CPU and system clock times for the various parts of the calculation (see Section 2.4.4).

FDC_inform is a scalar variable of type FDC_inform_type whose components are used to provide information about any detection of linear dependencies performed by the package GALAHAD_FDC. See the specification sheet for the package GALAHAD_FDC for details, and appropriate default values.

SBLS_inform is a scalar variable of type SBLS_inform_type whose components are used to provide information about factorizations performed by the package GALAHAD_SBLS. See the specification sheet for the package GALAHAD_SBLS for details, and appropriate default values.

2.4.6 The derived data type for holding problem data

The derived data type LSQP_data_type is used to hold all the data for a particular problem, or sequences of problems with the same structure, between calls of LSQP procedures. This data should be preserved, untouched, from the initial call to LSQP_initialize to the final call to LSQP_terminate.

2.5 Argument lists and calling sequences

There are three procedures for user calls (see Section 2.7 for further features):

1. The subroutine LSQP_initialize is used to set default values, and initialize private data, before solving one or more problems with the same sparsity and bound structure.
2. The subroutine LSQP_solve is called to solve the problem.
3. The subroutine LSQP_terminate is provided to allow the user to automatically deallocate array components of the private data, allocated by LSQP_solve, at the end of the solution process. It is important to do this if the data object is re-used for another problem **with a different structure** since LSQP_initialize cannot test for this situation, and any existing associated targets will subsequently become unreachable.

We use square brackets [ ] to indicate OPTIONALarguments.

2.5.1 The initialization subroutine

Default values are provided as follows:

```
CALL LSQP_initialize( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type LSQP_data_type (see Section 2.4.6). It is used to hold data about the problem being solved.

control is a scalar INTENT(OUT) argument of type LSQP_control_type (see Section 2.4.3). On exit, control contains default values for the components as described in Section 2.4.3. These values should only be changed after calling LSQP_initialize.

inform is a scalar INTENT(INOUT) argument of type LSQP_inform_type (see Section 2.4.5). A successful call to LSQP_initialize is indicated when the component status has the value 0. For other return values of status, see Section 2.6.

2.5.2 The linear or separable convex quadratic programming problem solution subroutine

The constrained least-distance problem solution algorithm is called as follows:

```
CALL LSQP_solve( p, data, control, inform[, C_stat, B_stat] )
```

p is a scalar INTENT(INOUT) argument of type QPT_problem_type (see Section 2.4.2). It is used to hold data about the problem being solved. For a new problem, the user must allocate all the array components, and set values for all components except p%C. p%new_problem_structure must be set .TRUE., but will have been reset to .FALSE. on exit from LSQP_solve. Users are free to choose whichever of the three matrix formats described in Section 2.1 is appropriate for **A** for their application.

For a problem with the same structure as one that has just been solved, the user may set p%new_problem_structure to .FALSE., so long as LSQP_terminate has not been called in the interim. The INTEGER(ip_) components must be unaltered since the previous call to LSQP_solve, but the REAL(rp_) may be altered to reflect the new problem.

The components p%X, p%Y and p%Z must be set to initial estimates, $\mathbf{x}^0$, of the primal variables, **x**, Lagrange multipliers for the general constraints, **y**, and dual variables for the bound constraints, **z**, respectively. Inappropriate initial values will be altered, so the user should not be overly concerned if suitable values are not apparent, and may be content with merely setting p%X=0.0, p%Y=0.0 and p%Z=0.0. The component p%C need not be set on entry.

On exit, the components p%X, p%Y, p%Z and p%C will contain the best estimates of the primal variables **x**, Lagrange multipliers for the general constraints **y**, dual variables for the bound constraints **z**, and values of the constraints **Ax** respectively. What of the remaining problem data has been restored depends upon the input value of the control parameter control%restore_problem. The return format for a restored array component will be the same as its input format. **Restrictions:** p%n >0, p%m ≥ 0 and p%A_ne ≥ -2.

data is a scalar INTENT(INOUT) argument of type LSQP_data_type (see Section 2.4.6). It is used to hold data about the problem being solved. It must not have been altered **by the user** since the last call to LSQP_initialize.

control is a scalar INTENT(IN) argument of type LSQP_control_type (see Section 2.4.3). Default values may be assigned by calling LSQP_initialize prior to the first call to LSQP_solve.

inform is a scalar INTENT(INOUT) argument of type LSQP_inform_type (see Section 2.4.5). A successful call to LSQP_solve is indicated when the component status has the value 0. For other return values of status, see Section 2.6.

C_stat is an OPTIONAL rank-one INTENT(OUT) array argument of dimension p%m and type INTEGER(ip_), that if PRESENT indicates which of the general linear constraints are likely in the optimal working set (that is a set of active constraints with linearly independent gradients). Possible values for C_stat(i), i$=1,\ldots,$ p%m, and their meanings are

- <0 the i-th general constraint is in the working set, on its lower bound,
- >0 the i-th general constraint is in the working set, on its upper bound, and
- 0 the i-th general constraint is not in the working set.

B_stat is an OPTIONAL rank-one INTENT(OUT) array argument of dimension p%n and type INTEGER(ip_), that that if PRESENT indicates which of the simple bound constraints are likely in the optimal working set. Possible values for B_stat(j), j$=1,\ldots,$ p%n, and their meanings are

- <0 the j-th simple bound constraint is in the working set, on its lower bound,
- >0 the j-th simple bound constraint is in the working set, on its upper bound, and
- 0 the j-th simple bound constraint is not in the working set.

2.5.3 The termination subroutine

All previously allocated arrays are deallocated as follows:

```
CALL LSQP_terminate( data, control, inform )
```

data is a scalar INTENT(INOUT) argument of type LSQP_data_type exactly as for LSQP_solve, which must not have been altered **by the user** since the last call to LSQP_initialize. On exit, array components will have been deallocated.

control is a scalar INTENT(IN) argument of type LSQP_control_type exactly as for LSQP_solve.

inform is a scalar INTENT(OUT) argument of type LSQP_inform_type exactly as for LSQP_solve. Only the component status will be set on exit, and a successful call to LSQP_terminate is indicated when this component status has the value 0. For other return values of status, see Section 2.6.

2.6 Warning and error messages

A negative value of inform%status on exit from LSQP_solve or LSQP_terminate indicates that an error has occurred. No further calls should be made until the error has been corrected. Possible values are:

-1. An allocation error occurred. A message indicating the offending array is written on unit control%error, and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc respectively.

-2. A deallocation error occurred. A message indicating the offending array is written on unit control%error and the returned allocation status and a string containing the name of the offending array are held in inform%alloc_status and inform%bad_alloc respectively. status is given by the value inform%alloc_status.

-3. One of the restrictions prob%n > 0 or prob%m ≥ 0 or requirements that prob%A_type contains its relevant string 'DENSE', 'COORDINATE' or 'SPARSE_BY_ROWS' has been violated.

-4. The bound constraints are inconsistent.

-5. The constraints appear to have no feasible point.

-7. The problem is unbounded from below. This can only happen if one (or more) $w_i = 0$ and its corresponding $g_i \neq 0$.

-8. The analytic center appears to be unbounded.

-9. The analysis phase of the factorization failed; the return status from the factorization package is given in the component inform%factor_status.

-10. The factorization failed; the return status from the factorization package is given in the component inform%factor_status.

-11. The solution of a set of linear equations using factors from the factorization package failed; the return status from the factorization package is given in the component inform%factor_status.

-16. The problem is so ill-conditioned that further progress is impossible.

-17. The step is too small to make further impact.

-18. Too many iterations have been performed. This may happen if control%maxit is too small, but may also be symptomatic of a badly scaled problem.

-19. The elapsed CPU or system clock time limit has been reached. This may happen if either control%cpu_time_limit or control%clock_time_limit is too small, but may also be symptomatic of a badly scaled problem.

2.7 Further features

In this section, we describe an alternative means of setting control parameters, that is components of the variable control of type LSQP_control_type (see Section 2.4.3), by reading an appropriate data specification file using the subroutine LSQP_read_specfile. This facility is useful as it allows a user to change LSQP control parameters without editing and recompiling programs that call LSQP.

A specification file, or specfile, is a data file containing a number of "specification commands". Each command occurs on a separate line, and comprises a "keyword", which is a string (in a close-to-natural language) used to identify a control parameter, and an (optional) "value", which defines the value to be assigned to the given control parameter. All keywords and values are case insensitive, keywords may be preceded by one or more blanks but values must not contain blanks, and each value must be separated from its keyword by at least one blank. Values must not contain more

than 30 characters, and each line of the specfile is limited to 80 characters, including the blanks separating keyword and value.

The portion of the specification file used by LSQP_read_specfile must start with a "BEGIN LSQP" command and end with an "END" command. The syntax of the specfile is thus defined as follows:

```
( .. lines ignored by LSQP_read_specfile .. )
  BEGIN LSQP
     keyword    value
     .......    .....
     keyword    value
  END
( .. lines ignored by LSQP_read_specfile .. )
```

where keyword and value are two strings separated by (at least) one blank. The "BEGIN LSQP" and "END" delimiter command lines may contain additional (trailing) strings so long as such strings are separated by one or more blanks, so that lines such as

```
BEGIN LSQP SPECIFICATION
```

and

```
END LSQP SPECIFICATION
```

are acceptable. Furthermore, between the "BEGIN LSQP" and "END" delimiters, specification commands may occur in any order. Blank lines and lines whose first non-blank character is ! or * are ignored. The content of a line after a ! or * character is also ignored (as is the ! or * character itself). This provides an easy manner to "comment out" some specification commands, or to comment specific values of certain control parameters.

The value of a control parameters may be of three different types, namely integer, logical or real. Integer and real values may be expressed in any relevant Fortran integer and floating-point formats (respectively). Permitted values for logical parameters are "ON", "TRUE", ".TRUE.", "T", "YES", "Y", or "OFF", "NO", "N", "FALSE", ".FALSE." and "F". Empty values are also allowed for logical control parameters, and are interpreted as "TRUE".

The specification file must be open for input when LSQP_read_specfile is called, and the associated device number passed to the routine in device (see below). Note that the corresponding file is REWINDed, which makes it possible to combine the specifications for more than one program/routine. For the same reason, the file is not closed by LSQP_read_specfile.

Control parameters corresponding to the components FDC_control and SBLS_control may be changed by including additional sections enclosed by "BEGIN FDC" and "END FDC", and "BEGIN SBLS" and "END SBLS", respectively. See the specification sheets for the packages GALAHAD_FDC and GALAHAD_SBLS for further details.

2.7.1 To read control parameters from a specification file

Control parameters may be read from a file as follows:

```
CALL LSQP_read_specfile( control, device )
```

control is a scalar INTENT(INOUT) argument of type LSQP_control_type (see Section 2.4.3). Default values should have already been set, perhaps by calling LSQP_initialize. On exit, individual components of control may have been changed according to the commands found in the specfile. Specfile commands and the component (see Section 2.4.3) of control that each affects are given in Table 2.1.

device is a scalar INTENT(IN) argument of type INTEGER(ip_), that must be set to the unit number on which the specfile has been opened. If device is not open, control will not be altered and execution will continue, but an error message will be printed on unit control%error.

command	component of control	value type
error-printout-device	%error	integer
printout-device	%out	integer
print-level	%print_level	integer
maximum-number-of-iterations	%maxit	integer
start-print	%start_print	integer
stop-print	%stop_print	integer
maximum-poor-iterations-before-infeasible	%infeas_max	integer
barrier-fixed-until-iteration	%muzero_fixed	integer
indicator-type-used	%indicator_type	integer
restore-problem-on-output	%restore_problem	integer
infinity-value	%infinity	real
primal-accuracy-required	%stop_p	real
dual-accuracy-required	%stop_d	real
complementary-slackness-accuracy-required	%stop_c	real
mininum-initial-primal-feasibility	%prfeas	real
mininum-initial-dual-feasibility	%dufeas	real
initial-barrier-parameter	%muzero	real
poor-iteration-tolerance	%reduce_infeas	real
minimum-potential-before-unbounded	%potential_unbounded	real
identical-bounds-tolerance	%identical_bounds_tol	real
primal-indicator-tolerance	%indicator_tol_p	real
primal-dual-indicator-tolerance	%indicator_tol_pd	real
tapia-indicator-tolerance	%indicator_tol_tapia	real
maximum-cpu-time-limit	%cpu_time_limit	real
maximum-clock-time-limit	%clock_time_limit	real
remove-linear-dependencies	%remove_dependencies	logical
treat-zero-bounds-as-general	%treat_zero_bounds_as_general	logical
just-find-feasible-point	%just_feasible	logical
get-advanced-dual-variables	%getdua	logical
move-final-solution-onto-bound	%feasol	logical

Table 2.1: Specfile commands and associated components of control.

2.8 Information printed

If control%print_level is positive, information about the progress of the algorithm will be printed on unit control%out. If control%print_level $= 1$, a single line of output will be produced for each iteration of the process. This will include values of the current primal and dual infeasibility, and violation of complementary slackness, the feasibility-phase objective value, the current steplength, the value of the barrier parameter, the number of backtracks in the linesearch and the elapsed clock time in seconds.

If control%print_level ≥ 2 this output will be increased to provide significant detail of each iteration. This extra output includes residuals of the linear systems solved, and, for larger values of control%print_level, values of the primal and dual variables and Lagrange multipliers.

3 GENERAL INFORMATION

Use of common: None.

Workspace: Provided automatically by the module.

Other routines called directly: None.

Other modules used directly: LSQP_solve calls the GALAHAD packages GALAHAD_CLOCK, GALAHAD_SYMBOLS, GALAHAD_SPACE, GALAHAD_TOOLS, GALAHAD_SPECFILE, GALAHAD_SMT, GALAHAD_QPT, GALAHAD_QPP, GALAHAD_QPD, GALAHAD_ROOTS, GALAHAD_SBLS and GALAHAD_FDC.

Input/output: Output is under control of the arguments control%error, control%out and control%print_level.

Restrictions: prob%n > 0, prob%m ≥ 0, prob%A_type $\in$ {'DENSE', 'COORDINATE', 'SPARSE_BY_ROWS'}.

Portability: ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

4 METHOD

The required solution $\mathbf{x}$ necessarily satisfies the primal optimality conditions

$$\mathbf{A}\mathbf{x} = \mathbf{c} \tag{4.1}$$

and

$$\mathbf{c}^l \leq \mathbf{c} \leq \mathbf{c}^u, \ \mathbf{x}^l \leq \mathbf{x} \leq \mathbf{x}^u, \tag{4.2}$$

the dual optimality conditions

$$\mathbf{W}^2(\mathbf{x}-\mathbf{x}^0)+\mathbf{g} = \mathbf{A}^T\mathbf{y}+\mathbf{z}, \ \mathbf{y} = \mathbf{y}^l+\mathbf{y}^u \text{ and } \mathbf{z} = \mathbf{z}^l+\mathbf{z}^u, \tag{4.3}$$

and

$$\mathbf{y}^l \geq 0, \ \mathbf{y}^u \leq 0, \ \mathbf{z}^l \geq 0 \text{ and } \mathbf{z}^u \leq 0, \tag{4.4}$$

and the complementary slackness conditions

$$(\mathbf{A}\mathbf{x}-\mathbf{c}^l)^T\mathbf{y}^l = 0, \ (\mathbf{A}\mathbf{x}-\mathbf{c}^u)^T\mathbf{y}^u = 0, \ (\mathbf{x}-\mathbf{x}^l)^T\mathbf{z}^l = 0 \text{ and } (\mathbf{x}-\mathbf{x}^u)^T\mathbf{z}^u = 0, \tag{4.5}$$

where the diagonal matrix $\mathbf{W}^2$ has diagonal entries w_j^2, $j = 1,\ldots,n$, the vectors $\mathbf{y}$ and $\mathbf{z}$ are known as the Lagrange multipliers for the general linear constraints, and the dual variables for the bounds, respectively, and where the vector inequalities hold componentwise.

Primal-dual interior point methods iterate towards a point that satisfies these conditions by ultimately aiming to satisfy (4.1), (4.3) and (4.5), while ensuring that (4.2) and (4.4) are satisfied as strict inequalities at each stage.

Appropriate norms of the amounts by which (4.1), (4.3) and (4.5) fail to be satisfied are known as the primal and dual infeasibility, and the violation of complementary slackness, respectively. The fact that (4.2) and (4.4) are satisfied as strict inequalities gives such methods their other title, namely interior-point methods.

When $\mathbf{w} \neq 0$ or $\mathbf{g} \neq 0$, the method aims at each stage to reduce the overall violation of (4.1), (4.3) and (4.5), rather than reducing each of the terms individually. Given an estimate $\mathbf{v} = (\mathbf{x}, \mathbf{c}, \mathbf{y}, \mathbf{y}^l, \mathbf{y}^u, \mathbf{z}, \mathbf{z}^l, \mathbf{z}^u)$ of the primal-dual variables, a correction $\Delta\mathbf{v} = \Delta(\mathbf{x}, \mathbf{c}, \mathbf{y}, \mathbf{y}^l, \mathbf{y}^u, \mathbf{z}, \mathbf{z}^l, \mathbf{z}^u)$ is obtained by solving a suitable linear system of Newton equations for the nonlinear systems (4.1), (4.3) and a parameterized perturbation of (4.5). An improved estimate $\mathbf{v} + \alpha\Delta\mathbf{v}$ is then used, where the stepsize α is chosen as close to 1.0 as possible while ensuring both that (4.2) and (4.4) continue to hold and that the individual components which make up the complementary slackness (4.5) do not deviate too significantly from their average value. The parameter that controls the perturbation of (4.5) is ultimately driven to zero. The Newton equations are solved by applying the GALAHAD matrix factorization package `GALAHAD_SBLS`, but there are options to factorize the matrix as a whole (the so-called "augmented system" approach), to perform a block elimination first (the "Schur-complement" approach), or to let the method itself decide which of the two previous options is more appropriate. The "Schur-complement" approach is usually to be prefered when all the weights are nonzero or when every variable is bounded (at least one side), but may be inefficient if any of the columns of $\mathbf{A}$ is too dense.

When $\mathbf{w} = 0$ and $\mathbf{g} = 0$, the method aims instead firstly to find an interior primal feasible point, that is to ensure that (4.1) is satisfied. One this has been achieved, attention is switched to mninizing the potential function

$$\phi(\mathbf{x}, \mathbf{c}) = -\sum_{i=1}^{m} \log(c_i - c_i^l) - \sum_{i=1}^{m} \log(c_i^u - c_i) - \sum_{j=1}^{n} \log(x_j - x_j^l) - \sum_{j=1}^{n} \log(x_j^u - x_j),$$

while ensuring that (4.1) remain satisfied and that $\mathbf{x}$ and $\mathbf{c}$ are strictly interior points for (4.2). The global minimizer of this minimization problem is known as the analytic center of the feasible region, and may be viewed as a feasible point that is as far from the boundary of the constraints as possible. Note that terms in the above sumations corresponding to infinite bounds are ignored, and that equality constraints are treated specially. Appropriate "primal" Newton corrections are used to generate a sequence of improving points converging to the analytic center, while the iteration is stabilized by performing linesearches along these corrections with respect to $\phi(\mathbf{x}, \mathbf{c})$.

In order to make the solution as efficient as possible, the variables and constraints are reordered internally by the GALAHAD package `GALAHAD_QPP` prior to solution. In particular, fixed variables, and free (unbounded on both sides) constraints are temporarily removed.

References:

The basic algorithm is that of

Y. Zhang (1994). On the convergence of a class of infeasible interior-point methods for the horizontal linear complementarity problem. SIAM J. Optimization **4** (1) 208-227,

with a number of enhancements described by

A. R. Conn, N. I. M. Gould, D. Orban and Ph. L. Toint (1999). A primal-dual trust-region algorithm for minimizing a non-convex function subject to general inequality and linear equality constraints. Mathematical Programming **87** 215-249.

5 EXAMPLE OF USE

Suppose we wish to find a point "closest" to $\mathbf{x}^0 = (-2, 1, 3)^T$ that satisfies the general linear constraints $1 \leq 2x_1 + x_2 \leq 2$, $x_2 + x_3 = 2$, and simple bounds $-1 \leq x_1 \leq 1$ and $x_3 \leq 2$. Suppose furthermore, that we wish to measure "closest" using firstly the weights $\mathbf{w} = (0.1, 1, 2)^T$ and secondly $\mathbf{w} = (1, 1, 1)^T$, and that we also wish to find the analytic center

of the feasible region. Then, on writing the data for this problem as

$$\mathbf{A} = \begin{pmatrix} 2 & 1 & \\ & 1 & 1 \end{pmatrix}, \ \mathbf{c}^l = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \ \mathbf{c}^u = \begin{pmatrix} 2 \\ 2 \end{pmatrix}, \ \mathbf{x}^l = \begin{pmatrix} -1 \\ -\infty \\ -\infty \end{pmatrix} \text{ and } \mathbf{x}^u = \begin{pmatrix} 1 \\ \infty \\ 2 \end{pmatrix}$$

we may use the following code—note that we ask for high accuracy when finding the analytic center by setting `control%stop_c` $= 10^{-16}$ and `control%itref_max` $= 2$:

```
! THIS VERSION: GALAHAD 2.2 - 23/04/2008 AT 16:30 GMT.
   PROGRAM GALAHAD_LSQP_EXAMPLE
   USE GALAHAD_LSQP_double                        ! double precision version
   IMPLICIT NONE
   INTEGER, PARAMETER :: wp = KIND( 1.0D+0 ) ! set precision
   REAL ( KIND = wp ), PARAMETER :: infinity = 10.0_wp ** 20
   TYPE ( QPT_problem_type ) :: p
   TYPE ( LSQP_data_type ) :: data
   TYPE ( LSQP_control_type ) :: control
   TYPE ( LSQP_inform_type ) :: inform
   INTEGER, PARAMETER :: n = 3, m = 2, a_ne = 4
   INTEGER :: i, s
! start problem data
   ALLOCATE( p%X_l( n ), p%X_u( n ) )
   ALLOCATE( p%C( m ), p%C_l( m ), p%C_u( m ) )
   ALLOCATE( p%X( n ), p%Y( m ), p%Z( n ) )
   p%new_problem_structure = .TRUE.           ! new structure
   p%n = n ; p%m = m ; p%f = 0.0_wp           ! dimensions & objective constant
   p%C_l = (/ 1.0_wp, 2.0_wp /)               ! constraint lower bound
   p%C_u = (/ 2.0_wp, 2.0_wp /)               ! constraint upper bound
   p%X_l = (/ - 1.0_wp, - infinity, - infinity /) ! variable lower bound
   p%X_u = (/ 1.0_wp, infinity, 2.0_wp /)     ! variable upper bound
   p%gradient_kind = 0
! sparse co-ordinate storage format: integer components
   CALL SMT_put( p%A%type, 'COORDINATE', s )  ! storage for H and A
   ALLOCATE( p%A%val( a_ne ), p%A%row( a_ne ), p%A%col( a_ne ) )
   p%A%row = (/ 1, 1, 2, 2 /)                 ! Jacobian A
   p%A%col = (/ 1, 2, 2, 3 /) ; p%A%ne = a_ne
! integer components complete
   CALL LSQP_initialize( data, control, inform ) ! Initialize control parameters
   control%infinity = infinity                   ! Set infinity
   control%restore_problem = 1                   ! Restore vector data on exit
!  control%print_level = 1
!  control%SBLS_control%symmetric_linear_solver = 'ma57'
!  control%SBLS_control%print_level = 1
!  control%FDC_control%print_level = 1
   DO i = 0, 2
!  DO i = 0, 1
     p%X = (/  -2.0_wp, 1.0_wp,  3.0_wp /)         ! set x0
     p%Y = 0.0_wp ; p%Z = 0.0_wp                   ! start multipliers from zero
! sparse co-ordinate storage format: real components
     p%A%val = (/ 2.0_wp, 1.0_wp, 1.0_wp, 1.0_wp /) ! Jacobian A
! real components complete
     p%Hessian_kind = 2 - i
     IF ( p%Hessian_kind == 0 ) THEN
       control%stop_c = 10.0_wp ** ( - 12 ) ; control%itref_max = 2
     END IF
```

```
    IF ( p%Hessian_kind == 2 ) THEN
      ALLOCATE( p%WEIGHT( n ) ) ; p%WEIGHT = (/ 0.1_wp, 1.0_wp, 2.0_wp /)
      ALLOCATE( p%X0( n ) )
    END IF
    IF ( p%Hessian_kind /= 0 ) p%X0 = p%X
    CALL LSQP_solve( p, data, control, inform )   ! Solve problem
    IF ( inform%status == 0 ) THEN                ! Successful return
      IF ( p%Hessian_kind == 0 ) THEN
        WRITE( 6, "( ' Eg ', I1, I6, ' iterations. Optimal potential value =',&
       &       ES12.4, /, ' Analytic center  = ', ( 5ES12.4 ) )" )            &
        i + 1, inform%iter, inform%potential, p%X
      ELSE
        WRITE( 6, "( ' Eg ', I1, I6, ' iterations. Optimal objective value =',&
       &       ES12.4, /, ' Optimal solution = ', ( 5ES12.4 ) )" )            &
        i + 1, inform%iter, inform%obj, p%X
      END IF
    ELSE                                          !  Error returns
      WRITE( 6, "( ' LSQP_solve exit status = ', I6 ) " ) inform%status
    END IF
  END DO
  CALL LSQP_terminate( data, control, inform )    !  delete internal workspace
  END PROGRAM GALAHAD_LSQP_EXAMPLE
```

This produces the following output:

```
 Eg 1     6 iterations. Optimal objective value =  2.5313E+00
 Optimal solution =   5.0022E-01  4.1431E-07  2.0000E+00
 Eg 2     8 iterations. Optimal objective value =  2.7500E+00
 Optimal solution =  -4.9773E-01  2.4977E+00  2.0000E+00
 Eg 3     7 iterations. Optimal potential value =  7.1493E-01
 Analytic center  =  -3.7381E-01  2.3013E+00 -3.0132E-01
```

The same problem may be solved holding the data in a sparse row-wise storage format by replacing the lines

```
! sparse co-ordinate storage format: integer components
...
! integer components complete
```

by

```
! sparse row-wise storage format: integer components
   CALL SMT_put( p%A%type, 'SPARSE_BY_ROWS' )  ! Specify sparse-by-row storage
   ALLOCATE( p%A%val( a_ne ), p%A%col( a_ne ), p%A%ptr( m + 1 )  )
   p%A%col = (/ 1, 2, 2, 3 /)                  ! Jacobian A
   p%A%ptr = (/ 1, 3, 5 /)                     ! Set row pointers
! integer components complete
```

and

```
! sparse co-ordinate storage format: real components
...
! real components complete
```

by

```
! sparse row-wise storage format: real components
     p%A%val = (/ 2.0_wp, 1.0_wp, 1.0_wp, 1.0_wp /) ! Jacobian A
! real components complete
```

or using a dense storage format with the replacement lines

```
! dense storage format: integer components
   CALL SMT_put( p%A%type, 'DENSE' )  ! Specify dense storage for A
   ALLOCATE( p%A%val( n * m ) )
! integer components complete
```

and

```
! dense storage format: real components
     p%A%val = (/ 2.0_wp, 1.0_wp, 0.0_wp, 0.0_wp, 1.0_wp, 1.0_wp /) ! Jacobian
! real components complete
```

respectively.